ASC Portal Design & Architecture

A Case Study in Grid Portal Development

Michael Paul Russell
Dept of Computer Science
The University of Chicago
russell@cs.uchicago.edu

Astrophysics Simulation Collaboratory

A Laboratory for Large Scale Simulations of Relativistic Astrophysics

A “Knowledge & Distributed Information” (KDI) Project
Funded by the National Science Foundation

Building a computational collaboratory to bring the numerical treatment of Einstein theory of general relativity to astrophysics

Principal investigators

Washington University	**Wai-Mo Suen**
Albert Einstein Institute	**Ed Seidel**
University of Chicago	**Ian Foster**
University of Illinois at Urbana-Champaign	**Mike Norman, John Shalf**
Rutgers University	**Manish Parashar**

Project goals

- Develop a useful software base for applying numerical treatment of Einstein theory to astrophysics research.
- Promote collaborative development among distributed teams of scientists, researchers and developers.
- To make our software and tools widely-available to the a astrophysics community.
- To coordinate the use of our software and computing resources among members of the astrophysics community.

Proposed solution

- An N-tier Web-based application environment for developing, distributing, and running Cactus and other useful applications on remote resources as well as tools for managing those resources.
- We'll use Grid technologies to overcome the practical obstacles of accessing resources due to the tremendous variety of resource types, mechanisms, and access policies that exist today.
- While Cactus provides the modular framework we require for building high-performance, parallelized, astrophysics applications that will compile and run in most computing environments.

Key advantages to thin clients

- We want to deliver most of our application services with DTHML based applications. Turns out with DHTML we can provide fairly sophisticated client-side behavior, it just takes a lot of work!

- But this means users can access most of our application services with any computer that has NS4+ & IE4+ installed. No other client-side configuration is required.

- Easy to introduce new application services or re-implement application services as required. Again, no need to reconfigure Internet browser or other client-side software.

Key advantages to application server

- Better able to coordinate user activity. For example may group user operations into more simple task units or introduce fault tolerance for critical operations.
- Can monitor user activity and implement reporting facilities, such emailing a user when a task completes or producing a weekly summary of user tasks.
- Can introduce collaborative tools such as online chats and means for users to share access to information and resources amongst each other.

Basic security requirements

- Maintain secure HTTP communication between client applications and our Web services.
- Use General Security Infrastructure (GSI) based authentication and authorization wherever possible.
- Maintain an organizational MyProxy service to enable users to store & retrieve short-lived GSI proxy certificates with our Web services.

Advanced security requirements

- Ability to work with multiple GSI proxy certificates per user session in order to authenticate to GSI-based services with different CAs and/or distinguished name entries.
- Allow users to share access to various information and resources where possible.
- Access control lists for restricting access on a per user basis to various functions or resources our offered by our services.

Basic application requirements

- (GSI)FTP access to remote file systems.
- (GSI)SSH access to remote computer systems.
- (GSI)CVS access to shared code repositories.
- (GSI)LDAP access to GIIS/GRIS and other directory services.
- GRAM job submission and integrated with job monitoring tools.

Advanced application requirements

- Flexible task management facilities. For example, allow user to spawn a file transfer in the background and check the status of that transfer at a later time. Ability to cancel tasks where possible, get a report to when the task is completed, schedule, etc…

- Ability to create and save custom tasks. First step: Allow user to edit and save a particular GRAM request they may perform often. Later: Integrate a task composition facility such as ANT.

Resource management requirements

- Manage “machine” definitions that users may utilize through our Web services and restrict views of those definitions as required.
- Facilities for verifying “machine” definitions, with the ability to setup tests on various services and/or with various certificates.
- Basic search capabilities (white and yellow pages) for finding appropriate resources on machines, primarily for locating the most appropriate job queues.

Software management requirements

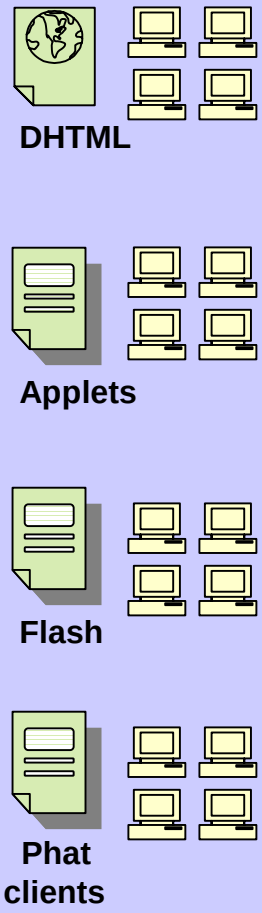
- Manage multiple Cactus distributions on remote machines.
- Checkout Cactus software from multiple CVS repositories into remote distributions.
- Build Cactus applications remotely and the ability to maintain configuration settings per machine to ease the build process.
- Edit and easily distribute Cactus parameter files.
- Run and monitor Cactus applications on remote machines.

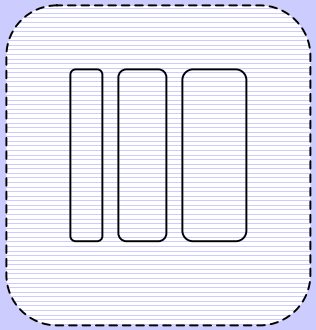
Basic administrative requirements

- Manage user accounts, edit their profiles, even to kill user sessions that may have been lost by the user.
- Monitor user activity (tasks), even to kill tasks where possible.
- Generate regular reports on user activity or anything useful for administrative, research, or funding purposes.

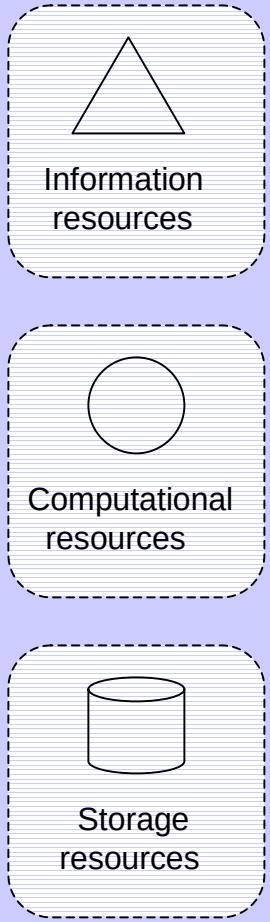
Grid-tier architecture

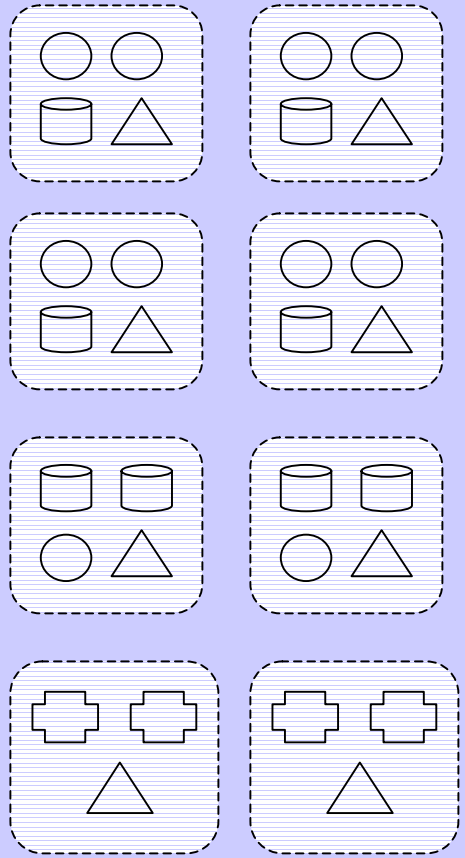
Remote clients	Application server	Resource brokers	Remote resources / Additional Services



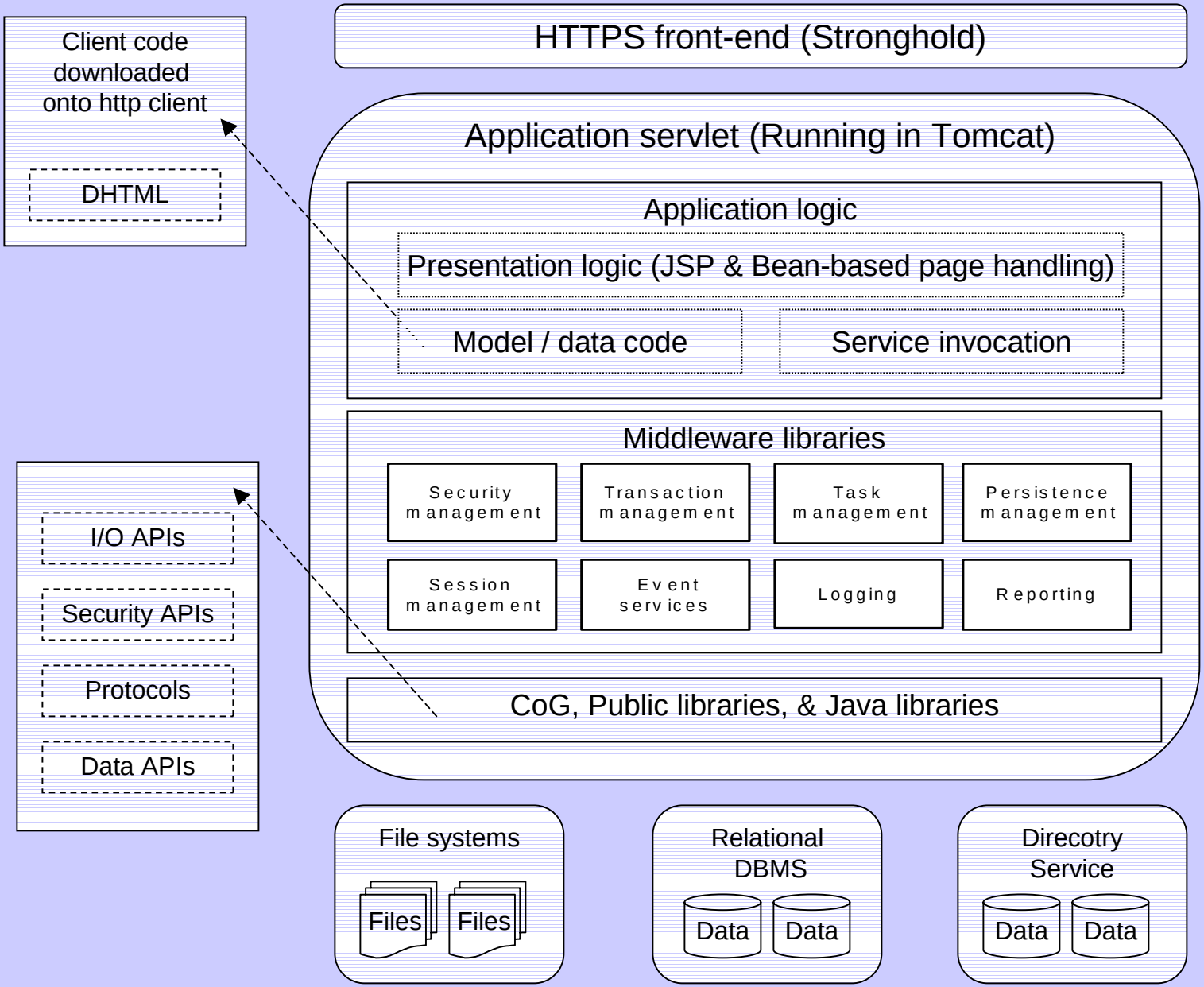


1. Application services
2. Local resource brokers
3. Local resources



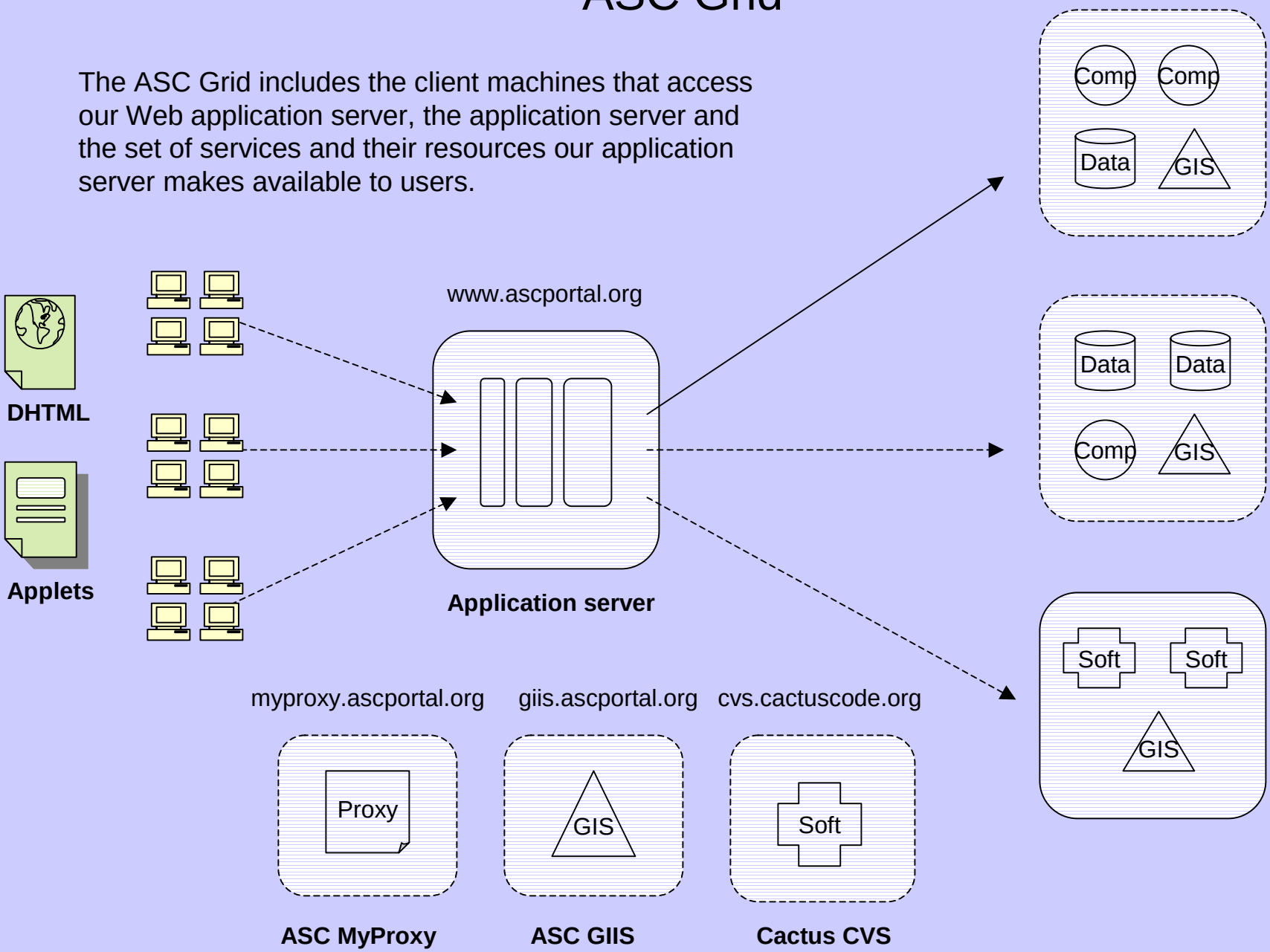


Server-side architecture



ASC Grid

The ASC Grid includes the client machines that access our Web application server, the application server and the set of services and their resources our application server makes available to users.



ASC Software

GridSphere (org.ascportal.gridsphere)

Basic application server infrastructure for building multi-user, multi-threaded application services for utilizing Grid technologies to gain access to remote resources.

- **Not** Servlet / JSP specific, want to be flexible.
- Utilizes Java CoG, introduces new extensions
- Data model **is a** part of this infrastructure
- JDBC based persistence (only, for now that is).

Orbiter (org.ascportal.gridsphere.orbiter)

JSP-based application service built upon GridSphere.

- Delivers DHTMTL client-side applications
- JSP-page infrastructure for constructing pages
- Page-oriented Java Beans handle application logic

GridSphere packages

org.ascportal.gridsphere [one day packaged under org.gridsphere?]

- **cactus** : code for working with Cactus on remote resources
- **grid** : extensions to Grid technologies
  - **clients** : incorporates machines and security packages with grid client protocols provided by Java CoG.
  - **machines** : extended machine management
  - **security** : extended credential management
  - **tasks** : encapsulates client requests as tasks
- **logging** : generic log management
- **modules** : code for managing application requests.
- **orbiter** : Orbiter packages
- **reporting** : generic report management
- **security** : application server security
- **task** : generic task management
- **user** : user profile and session management

Application server goals

- Develop an extensible Java application architecture that utilizes the latest in Java technologies, Grid technologies, and Internet standards for building multi-user, multi-threaded applications.
- Develop Java Servlet, RMI, CORBA, etc… extensions to this architecture.
- Design all functions and tasks such that they can be monitored and controlled by other users (administrators) if necessary.
- Since we track the state of remote files, jobs, etc., we need to maintain consistency, thus need mechanisms for syncing up with changes that are made externally.
- Fault tolerance… going beyond transaction management.

Client application goals

- Lots of thin-client design and applications ideas to pursue:
  - Bring coding standards to the browser. Companies invest tons of resources designing user interfaces, so should we… we are building **end-user** applications.
  - Bring the command line to the browser: Interactive shell access to remote computers (GSI-SSH based), as Java applet or even in DHTML. In my opinion, no operating environment is complete without both command-line interfaces and GUIs.

- Offline-client idea: Be able to setup tasks on offline with a client application and then sync up once back online, as with popular email programs like Eudora and Outlook. Another

General design concepts

- Want to build compelling and comprehensive environments so that users will want to conduct their daily activities with these environments.

- May be unnatural to classify portals as user, application, or administrative only. Instead be aware we're developing infrastructures in which communities will communicate and conduct research (business). There is a lot of management that will be necessary to maintain these services.

- Don't forget to account for these things in your design:
  - Grid services may not be available or properly configured on remote resources.
  - Someone needs to maintain all those services you've setup!
  - Should be able to administer all aspects of your application services at runtime if you care about high-availability